Benchmarking the pathway fingerprint - squared mean rank - hpc111 version

Gabriel Altschuler

September 21, 2011

In this document the ability of the fingerprint to classify samples across species will be evaluated. Two datasets will be used, tissue samples, and cell types in the hematopoiesis lineage.

Contents

1 Cross-species tissue identification **1**
1.1 Classification by nearest neighbor 3
1.1.1 comparison to Barcode 6
1.2 Intra- vs inter-tissue distance 6
1.3 K-fold cross validation 8
1.4 Precision-Recall 18
1.4.1 Approach 1 18
1.4.2 Approach 2 27
1.4.3 Comparison to randomly constructed genesets 34
1.4.4 Approach 3 39

2 Figure for manuscript **46**

3 Cross-species hematopoesis profiling **46**

1 Cross-species tissue identification

A list of tissue-specific datasets was taken from *Zilliox and Irizarry. A gene expression bar code for microarray data. Nat Meth (2007) vol. 4 (11) pp. 911-3.* This is composed of arrays from 6 tissues; muscle, lung, spleen, kidney, liver and brain.
First we need to load the probability of expression (POE) matrix that corresponds to this dataset, referred to as the *barcode dataset*, and the accompanying metadata. N.B. The POE matrix is the pathway data matrix prior to applying a threshold to produce the ternary fingerprint.

```
> library(pathprint)
> load(
+  "/home/galtschu2/Documents/Projects/Fingerprinting/data/sq_.POE.matrix.2011-07-27.RDa
+   )
> barcode.meta<-read.delim(
+     "/data/shared/Fingerprint/curatedCellTypes/barcode_figure2_data.txt",
+     stringsAsFactors = FALSE)
> barcode.meta<-barcode.meta[barcode.meta$DB_ID %in% GEO.metadata.matrix$GSM,]
```

We only use the samples that are in the matrix. The full barcode set includes yeast negative controls that are not processed by the fingerprint as they have the incorrect species identifier. In addition, there are arrays that were not completely uploaded to GEO and so were removed from the processing pipeline. The next step is to define a function to convert a POE matrix to a ternary matrix, according to a threshold

```
> ternaryThreshold <- function(matrix, threshold)
+   # function to convert a POE matrix to a thresholded matrix
+   {
+   high<-threshold
+   low<-(-threshold)
+   threshold.matrix<-(matrix>high)-(matrix<low)
+   threshold.matrix[is.na(threshold.matrix)]<-0
+   return(threshold.matrix)
+   }
> #N<-c(seq(0.001, 0.01, 0.001), seq(0.02, 0.1,0.01), seq(0.2, 0.5, 0.1), seq(1, 7, 1))
> N<-c(0.001, seq(0.0025, 0.01, 0.0025), seq(0.02, 0.1,0.01), seq(0.2, 0.5, 0.1), 0.75,
```

Now define separate matrices for each platform. There is no need to maintain the full POE matrix in memory after this, espcecially as the way in which the parallel processes are spawned later in the script would replicate this across all forks. N.B even with manual garbage collection this memory is not returned to the system until the R session is closed. This is possibly a function of the operating system.

```
> GPL570<-barcode.meta[barcode.meta$Platform == "GPL570",]
> GPL96<-barcode.meta[barcode.meta$Platform == "GPL96",]
> GPL1261<-barcode.meta[barcode.meta$Platform == "GPL1261",]
> GPL570.matrix<-POE.matrix[,GPL570$DB_ID]
> GPL96.matrix<-POE.matrix[,GPL96$DB_ID]
> GPL1261.matrix<-POE.matrix[,GPL1261$DB_ID]
> rm(POE.matrix)
> # Construct matrix combinations for later use
> human.matrix<-cbind(GPL570.matrix, GPL96.matrix)
> full.matrix<-cbind(human.matrix, GPL1261.matrix)
> human.data<-rbind(GPL570, GPL96)
> mouse.class<-GPL1261$Tissue
```

```
> human.class<-human.data$Tissue
> names(mouse.class)<-GPL1261$DB_ID
> names(human.class)<-human.data$DB_ID
> full.class<-c(human.class, mouse.class)

> load("~/Documents/Projects/Fingerprinting/data/random_sq_.POE.matrix.2011-08-08.RData"
> GPL570.random.matrix<-POE.matrix[,GPL570$DB_ID]
> GPL96.random.matrix<-POE.matrix[,GPL96$DB_ID]
> GPL1261.random.matrix<-POE.matrix[,GPL1261$DB_ID]
> human.random.matrix<-cbind(GPL570.random.matrix, GPL96.random.matrix)
> rm(POE.matrix)

> load("/data/shared/Fingerprint/misc/barcode.ranks.RData")
> barcode.ranks<-barcode.ranks[,names(full.class)]
> # remove genes with any NAs
> if (length(which(is.na(barcode.ranks))) > 0){
+   na.row<-which(is.na(barcode.ranks), arr.ind = TRUE)[,1]
+   barcode.ranks<-barcode.ranks[-na.row,]
+   }
```

1.1 Classification by nearest neighbor

We will now use a simple nearest-neighbor approach to identify the closest human array to each mouse array. This gives us the predicted tissue. We will cycle through a range of threshold values. For expediency this can be run this in parallel on the server using the package `muilticore`.

```
> library(multicore)
> library(doMC)
> # run over 15 cores - i.e. 2 thresholds per core
> registerDoMC(cores = 15)
> class <- foreach (j = 1:30) %dopar% {
+   GPL1261$predictedTissue<-NA
+   n <- N[j]
+   threshold<-10^(-n)
+   human.threshold<-ternaryThreshold(human.matrix, threshold)
+   GPL1261.threshold<-ternaryThreshold(GPL1261.matrix, threshold)
+   ordered<-vector("list", nrow(GPL1261))
+   for (i in 1:nrow(GPL1261)){
+     mouse.fingerprint<-GPL1261.threshold[,GPL1261$DB_ID[i], drop = FALSE]
+     ordered <- sort(colSums(abs(apply(
+                         human.threshold, 2, function(x){x - mouse.fingerprint}
+                                       ))))
+     GPL1261$predictedTissue[i]<-human.data$Tissue[
+                                     human.data$DB_ID == names(ordered)[1]
```

```
+                                    ]
+     }
+   Actual <- as.factor(GPL1261$Tissue)
+   Predicted<-factor(GPL1261$predictedTissue, levels = levels(Actual))
+   m <- table(Actual = Actual, Predicted = Predicted)
+   err <- c(threshold, 1 - sum(diag(m)) / sum(m))
+   return(list(err = err, m = m))
+ }
> m<-lapply(class, function(x){x[["m"]]})
> err<-t(sapply(class, function(x){x[["err"]]}))
```

The confusion matrix is used to calculate the error rate, as shown for a threshold of 0.001.

```
> m[[which(N == 3)]]
                 Predicted
Actual            brain kidney liver lung skeletal muscle spleen
  brain              39      0     0    0                0      0
  kidney              0     34     0    0                0      0
  liver               0      0   160    0                0      0
  lung                0      0     0   83                0      1
  skeletal muscle     0      0     0    0                2      0
  spleen              0      0     0    0                0     17
```

This table gives an error rate of

```
> 1 - sum(diag(m[[which(N == 3)]])) / sum(m[[which(N == 3)]])
[1] 0.002976190
```

```
> plot(err, log = "x", xlab = "Ternary threshold", ylab = "Error rate",
+     main = "Nearest neighbor, GPL1261 to GPL96")
```

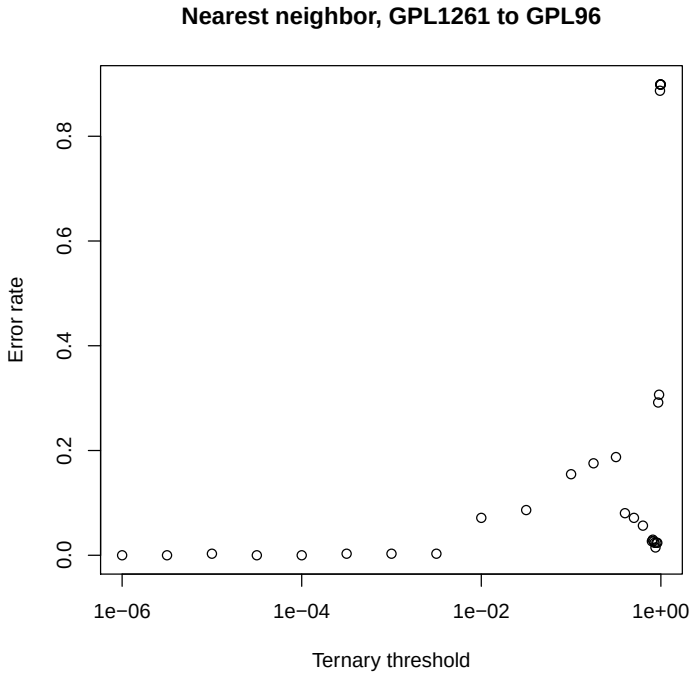

Figure 1: Error rate vs fingerprint threshold

1.1.1 comparison to Barcode

How does this compare to the barcode data? The barcode matrix based on common genes has already been compiled

```
> load("/data/shared/Fingerprint/misc/barcode.matrix.RData")
> GPL570.barcode.matrix<-barcode.matrix[,GPL570$DB_ID]
> GPL96.barcode.matrix<-barcode.matrix[,GPL96$DB_ID]
> GPL1261.barcode.matrix<-barcode.matrix[,GPL1261$DB_ID]
> human.barcode.matrix<-cbind(GPL570.barcode.matrix, GPL96.barcode.matrix)
> full.barcode.matrix<-cbind(human.barcode.matrix, GPL1261.barcode.matrix)
```

The confusion matrix can be calculated in the same way

```
> GPL1261$predictedTissueBarcode<-NA
> GPL1261$predictedTissueBarcode<-unlist(
+   foreach (i = 1:nrow(GPL1261)) %dopar% {
+   mouse.barcode<-barcode.matrix[,GPL1261$DB_ID[i], drop = FALSE]
+   ordered <- sort(colSums(abs(apply(human.barcode.matrix,
+                                     2,
+                                     function(x){x - mouse.barcode}
+                                     ))))
+   return(human.data$Tissue[human.data$DB_ID == names(ordered)[1]])
+   })
> table(Actual = GPL1261$Tissue, Predicted = GPL1261$predictedTissueBarcode)
                Predicted
Actual           brain kidney liver lung skeletal muscle spleen
  brain             39      0     0    0               0      0
  kidney             0     34     0    0               0      0
  liver              0      0   160    0               0      0
  lung               0      0     0   84               0      0
  skeletal muscle    0      0     0    0               2      0
  spleen             0      0     0    0               0     17
```

It seems that barcode produces perfect scores in this dataset.

1.2 Intra- vs inter-tissue distance

The ratio of the within-class to between-class variance can be used as another means to quantify the extent to which the fingerprints formed discrete tissue groups. Variance was defined as sum of the squared distance (Euclidean or Manhattan) between each sample and the mean pathway fingerprint for each tissue.

```
> varRatio<-function(matrix, colClasses, method = "manhattan"){
+   # check values
+   if (!(all.equal(colnames(matrix), names(colClasses)))
+         ) stop("Matrix colnames and class names should match")
```

```
+   class.levels<-levels(as.factor(colClasses))
+   class.means<-sapply(class.levels, function(x){
+     rowMeans(matrix[,names(colClasses)[colClasses == x]])
+     })
+   intra.var<-vector("numeric", length(class.levels))
+   names(intra.var) <- class.levels
+   inter.var.sum<-vector("numeric", length(class.levels))
+   names(inter.var.sum) <- class.levels
+   for (i in class.levels){
+     grp.1 <- names(colClasses)[colClasses == i]
+
+     if(method == "spearman"){
+           dist2 <- function(x, ...)
+             as.dist(1-cor(t(x), method="spearman"))
+           intra.var[i] <- sum((dist2(t(cbind(class.means[,i], matrix[,grp.1])),
+                       method = method)[1:ncol(matrix[,grp.1])]
+                   )^2)
+       } else{
+       intra.var[i] <- sum((dist(t(cbind(class.means[,i], matrix[,grp.1])),
+                       method = method)[1:ncol(matrix[,grp.1])]
+                    )^2)
+       }
+     inter.var <- vector("numeric", length(class.levels[class.levels != i]))
+     names(inter.var) <- class.levels[class.levels != i]
+     for (j in class.levels[class.levels != i]){
+     if(method == "spearman"){
+       inter.var[j] <- sum((dist2(t(cbind(class.means[,j], matrix[,grp.1])),
+                       method = method)[1:ncol(matrix[,grp.1])]
+                   )^2)
+     } else {
+       inter.var[j] <- sum((dist(t(cbind(class.means[,j], matrix[,grp.1])),
+                       method = method)[1:ncol(matrix[,grp.1])]
+                   )^2)
+     }
+     }
+     inter.var.sum[i] <- sum(inter.var)
+     }
+   ratio <- sum(intra.var)/sum(inter.var)
+   return(ratio)
+   }

> thresholds.varRatio.manhattan<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   varRatio(full.threshold, full.class, method = "manhattan")
```

```
+   })
> unthresholded.varRatio.manhattan<-varRatio(full.matrix,
+                                            full.class,
+                                            method = "manhattan")
> barcode.varRatio.manhattan<-varRatio(full.barcode.matrix,
+                                      full.class,
+                                      method = "manhattan")
> thresholds.varRatio.euclidean<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   varRatio(full.threshold, full.class, method = "euclidean")
+   })
> unthresholded.varRatio.euclidean<-varRatio(full.matrix,
+                                            full.class,
+                                            method = "euclidean")
> barcode.varRatio.euclidean<-varRatio(full.barcode.matrix,
+                                      full.class,
+                                      method = "euclidean")
> ranks.varRatio.spearman<-varRatio(barcode.ranks,
+                                      full.class,
+                                      method = "spearman")

> plot(x = 10^(-N),
+      y = unlist(thresholds.varRatio.manhattan),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Intra/Inter cluster variance",
+      ylim = c(0, 1)
+      )
> abline(h = unthresholded.varRatio.manhattan, lty = 2)

> plot(x = 10^(-N),
+      y = unlist(thresholds.varRatio.euclidean),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Intra/Inter cluster variance",
+      ylim = c(0, 1)
+      )
> abline(h = unthresholded.varRatio.euclidean,  lty = 2)
```

1.3 K-fold cross validation

The data sets can be divided into K subsets of equal, or approximately equal, size. One of the subsets is omitted and mean pathway fingerprints calculated

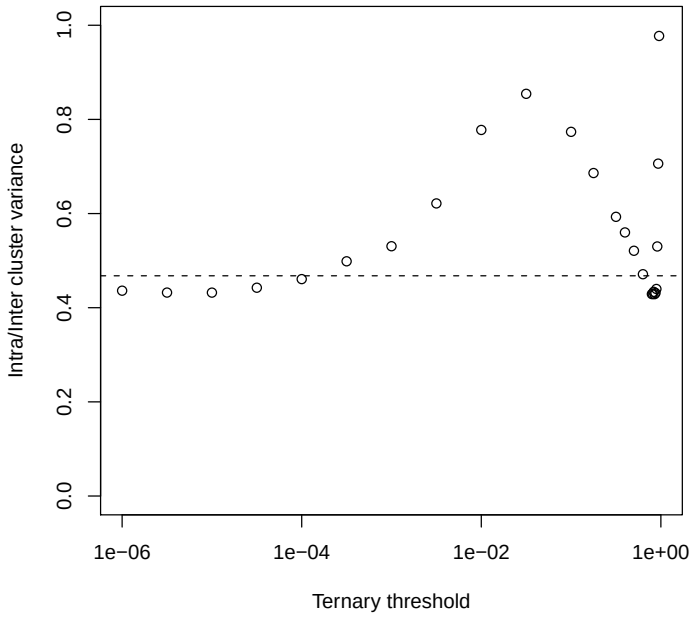

Figure 2: Plot of the intra-cluter vs inter-cluster variance ratio over the range of thresholds tested for the aggregated dataset. Variance is defined based on the Manhattan distance. Dashed line indicates the ratio for the unthresholded POE matrix, the barcode produces a ratio of 1.107, and gene expression (based on spearman rank) a ratio of 0.919. The fingerprint at 10E-3 produces a ratio of 0.531, and the unthresholded POE scores a ratio of 0.468.

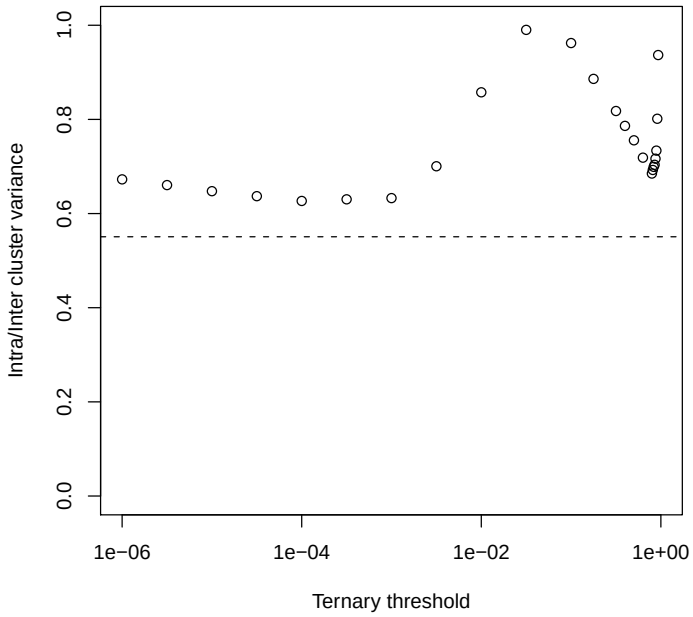

Figure 3: Plot of the intra-cluter vs inter-cluster variance ratio over the range of thresholds tested for the aggregated dataset. Variance is defined based on the Euclidean distance. Dashed line indicates the ratio for the unthresholded POE matrix, the barcode produces a ratio of 1.257, and gene expression (based on spearman rank) a ratio of 0.919. The fingerprint at 10E-3 produces a ratio of 0.633, and the unthresholded POE scores a ratio of 0.551.

for each tissue from the remaining samples. Next, the samples in the omitted subset are assigned to the tissue with the closest mean tissue pathway fingerprint (by Euclidean or Manhattan distance) and these assignments compared to the known annotations, from which an an error rate is calculated. This is repeated, omitting each of the subsets in turn, to obtain a mean error rate. This cross validation procedure is performed 10 times for each threshold value to estimate the mean and standard deviation of the error rate. This proceedure is performed here for K = 5 and K = sample size (i.e. leave-one-out cross validation).

```
> crossValidation<-function(matrix, colClasses, K = 5, method = "manhattan")
+   {
+   # check values
+   if (!(all.equal(colnames(matrix), names(colClasses)))) stop(
+     "Check test matrix and class names shouldmatch")
+   if ((K > length(colClasses)) | (K <=1)) stop(
+     "K outside allowable range")
+   # split into groups
+   n <- ncol(matrix)
+   K.o <- K
+   K <- round(K)
+   kvals <- unique(round(n/(1L:floor(n/2))))
+   temp <- abs(kvals - K)
+     if (!any(temp == 0))
+         K <- kvals[temp == min(temp)][1L]
+     if (K != K.o)
+         warning("K has been set to ", K)
+     f <- ceiling(n/K)
+     s <- sample(rep(1L:K, f), n)
+     n.s <- table(s)
+     ms <- max(s)
+     CV <- vector("numeric", length(seq_len(ms)))
+     for (i in seq_len(ms)) {
+         j.out <- seq_len(n)[(s == i)]
+         j.in <- seq_len(n)[(s != i)]
+         matrix.out <- matrix[,j.out, drop = FALSE]
+         matrix.in <- matrix[,j.in]
+         colClasses.out <- colClasses[colnames(matrix.out)]
+         colClasses.in <- colClasses[colnames(matrix.in)]
+         # Insert function here
+         class.levels.in<-levels(as.factor(colClasses.in))
+         class.means.in<-sapply(class.levels.in, function(x){
+           rowMeans(matrix.in[,names(colClasses.in)[colClasses.in == x]])
+             })
+         predictedTissue<-vector("character", ncol(matrix.out))
+         for (j in 1:ncol(matrix.out)){
+           if (method == "manhattan"){
```

```
+             predictedTissue[j] <- names(which.min(colSums(abs(apply(class.means.in,
+                           2,
+                           function(x){x - matrix.out[,j, drop = FALSE]}
+                           )))))
+             }
+         else if (method == "euclidean"){
+             predictedTissue[j] <- names(which.min(colSums(abs(apply(class.means.in,
+                           2,
+                           function(x){(x - matrix.out[,j, drop = FALSE])^2}
+                           )))))
+           }
+         }
+         Actual <- as.factor(colClasses.out)
+         Predicted<-factor(predictedTissue, levels = levels(Actual))
+         m <- table(Actual = Actual, Predicted = Predicted)
+         err <- (1 - sum(diag(m)) / sum(m))
+         CV[i]<-err
+     }
+   # NaN occurs when there are no correct entries, i.e. error rate is 1
+   CV[is.na(CV)] <- 1
+   CV.av<-mean(CV)
+   return(CV.av)
+ }

> # Manhattan distance
> fiveFold.manhattan<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   sapply(1:10, function(x){
+     crossValidation(matrix = full.threshold,
+     colClasses = full.class,
+     K = 5,
+     method = "manhattan")
+       }
+          )
+   })
> unthresholded.fiveFold.manhattan<-sapply(1:10, function(x){
+     crossValidation(matrix = full.matrix,
+     colClasses = full.class,
+     K = 5,
+     method = "manhattan")
+     }
+    )
> barcode.fiveFold.manhattan<-sapply(1:10, function(x){
+     crossValidation(matrix = full.barcode.matrix,
```

```
+     colClasses = full.class,
+     K = 5,
+     method = "manhattan")
+     }
+    )
> ranks.fiveFold.manhattan<-sapply(1:10, function(x){
+     crossValidation(matrix = barcode.ranks,
+     colClasses = full.class,
+     K = 5,
+     method = "manhattan")
+     }
+    )
> # Euclidean distance
> fiveFold.euclidean<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   sapply(1:10, function(x){
+     crossValidation(matrix = full.threshold,
+     colClasses = full.class,
+     K = 5,
+     method = "euclidean")
+       }
+          )
+   })
> unthresholded.fiveFold.euclidean<-sapply(1:10, function(x){
+     crossValidation(matrix = full.matrix,
+     colClasses = full.class,
+     K = 5,
+     method = "euclidean")
+     }
+    )
> barcode.fiveFold.euclidean<-sapply(1:10, function(x){
+     crossValidation(matrix = full.barcode.matrix,
+     colClasses = full.class,
+     K = 5,
+     method = "euclidean")
+     }
+    )
> ranks.fiveFold.euclidean<-sapply(1:10, function(x){
+     crossValidation(matrix = barcode.ranks,
+     colClasses = full.class,
+     K = 5,
+     method = "euclidean")
+     }
+    )
```

```
> library(gplots)
gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.
> plotCI(x = 10^(-N),
+      y = sapply(fiveFold.manhattan, mean),
+      uiw = sapply(fiveFold.manhattan, sd),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Mean error rate",
+      ylim = c(0, 1),
+      gap = 0,
+      main = "Manhattan Distance"
+      )
> abline(h = mean(unthresholded.fiveFold.manhattan), lty = 1)
> abline(h = mean(unthresholded.fiveFold.manhattan)+
+   sd(unthresholded.fiveFold.manhattan), lty = 2)
> abline(h = mean(unthresholded.fiveFold.manhattan)-
+   sd(unthresholded.fiveFold.manhattan), lty = 2)
> abline(h = mean(barcode.fiveFold.manhattan), lty = 1, col = "red")
> abline(h = mean(barcode.fiveFold.manhattan)+
+   sd(barcode.fiveFold.manhattan), lty = 2, col = "red")
> abline(h = mean(barcode.fiveFold.manhattan)-
+   sd(barcode.fiveFold.manhattan), lty = 2, col = "red")


> plotCI(x = 10^(-N),
+      y = sapply(fiveFold.euclidean, mean),
+      uiw = sapply(fiveFold.euclidean, sd),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Mean error rate",
+      ylim = c(0, 1),
+      gap = 0,
+      main = "Euclidean Distance"
+      )
> abline(h = mean(unthresholded.fiveFold.euclidean), lty = 1)
> abline(h = mean(unthresholded.fiveFold.euclidean)+
+   sd(unthresholded.fiveFold.euclidean), lty = 2)
> abline(h = mean(unthresholded.fiveFold.euclidean)-
+   sd(unthresholded.fiveFold.euclidean), lty = 2)
> abline(h = mean(barcode.fiveFold.euclidean), lty = 1, col = "red")
> abline(h = mean(barcode.fiveFold.euclidean)+
+   sd(barcode.fiveFold.euclidean), lty = 2, col = "red")
> abline(h = mean(barcode.fiveFold.euclidean)-
+   sd(barcode.fiveFold.euclidean), lty = 2, col = "red")
```

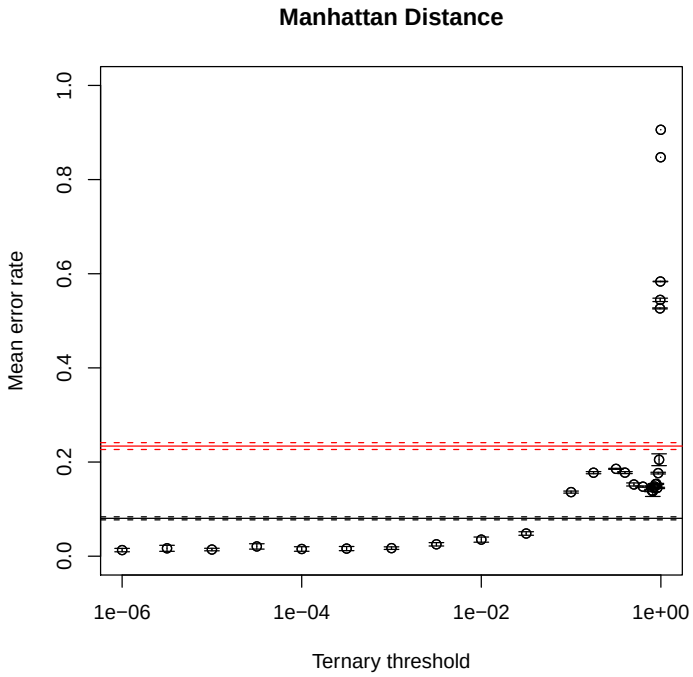

Figure 4: Plot of the average error rate based on 10 repeats of a 5-fold cross-validation over the range of thresholds tested for the aggregated dataset based on the Manhattan distance. Error bars indicate -/+ 1 std.dev. The black line indicates the ratio for the unthresholded POE matrix, and the red for the barcode, dashed lines indicate -/+ 1 std.dev. The barcode produces an error rate of 0.234, SD = 0.007, and gene expression (based on spearman rank) a rate of 0.64, SD = 0.01 (N.B. this part is based on distance rather than 1-correlation). The fingerprint at 10E-3 produces a ratio of 0.017, SD = 0.003, and the unthresholded POE scores a ratio of 0.08, SD = 0.003.

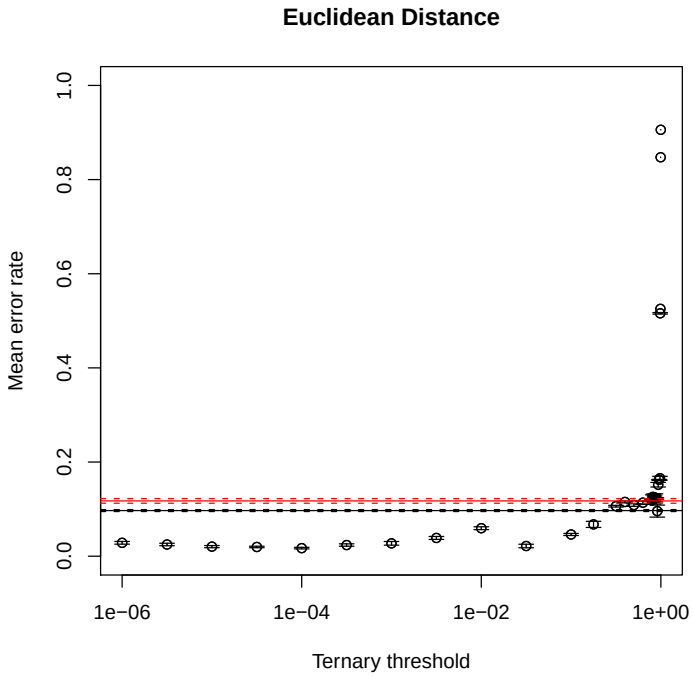

Figure 5: Plot of the average error rate based on 10 repeats of a 5-fold cross-validation over the range of thresholds tested for the aggregated dataset based on the Euclidean distance. Error bars indicate -/+ 1 std.dev. The black line indicates the ratio for the unthresholded POE matrix, and the red for the barcode, dashed lines indicate -/+ 1 std.dev. The barcode produces an error rate of 0.117, SD = 0.005, and gene expression (based on spearman rank) a rate of 0.633, SD = 0.003 (N.B. this part is based on distance rather than 1-correlation). The fingerprint at 10E-3 produces a ratio of 0.027, SD = 0.004, and the unthresholded POE scores a ratio of 0.097, SD = 0.002.

```
> # Manhattan distance
> LOO.manhattan<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   crossValidation(matrix = full.threshold,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "manhattan")
+   })
> unthresholded.LOO.manhattan<-
+     crossValidation(matrix = full.matrix,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "manhattan")
> barcode.LOO.manhattan<-
+     crossValidation(matrix = full.barcode.matrix,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "manhattan")
> ranks.LOO.manhattan<-
+     crossValidation(matrix = barcode.ranks,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "manhattan")
> # Euclidean distance
> LOO.euclidean<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   crossValidation(matrix = full.threshold,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "euclidean")
+   })
> unthresholded.LOO.euclidean<-
+     crossValidation(matrix = full.matrix,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "euclidean")
> barcode.LOO.euclidean<-
+     crossValidation(matrix = full.barcode.matrix,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "euclidean")
> ranks.LOO.euclidean<-
```

```
+     crossValidation(matrix = barcode.ranks,
+     colClasses = full.class,
+     K = length(full.class),
+     method = "euclidean")

> library(gplots)
> plotCI(x = 10^(-N),
+      y = sapply(LOO.manhattan, mean),
+      uiw = sapply(LOO.manhattan, sd),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Mean error rate",
+      ylim = c(0, 1),
+      gap = 0,
+      main = "Manhattan Distance")
> abline(h = unthresholded.LOO.manhattan, lty = 1)
> abline(h = barcode.LOO.manhattan, lty = 1, col = "red")

> plotCI(x = 10^(-N),
+      y = sapply(LOO.euclidean, mean),
+      uiw = sapply(LOO.euclidean, sd),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Mean error rate",
+      ylim = c(0, 1),
+      gap = 0,
+      main = "Euclidean Distance")
> abline(h = unthresholded.LOO.euclidean, lty = 1)
> abline(h = barcode.LOO.euclidean, lty = 1, col = "red")
```

1.4 Precision-Recall

Alternative means of benchmarking is to define precision-recall curves. We will adopt 4 approaches, 1 and 2 consider mouse to human matching, the first grouping all samples together, while the second considers each tissue class separately. In the third approach we will match across all samples, irrespective of species (i.e. not just mouse to human).

1.4.1 Approach 1

For this we will calculate the distance from each mouse array to all of the human arrays (or vice-versa). This can be used to construct a list of all the pairs, ranked by their distance. The precision and recall can be calculated at each step as we descend this ranked list.
First, let's define a function to output the precision and recall

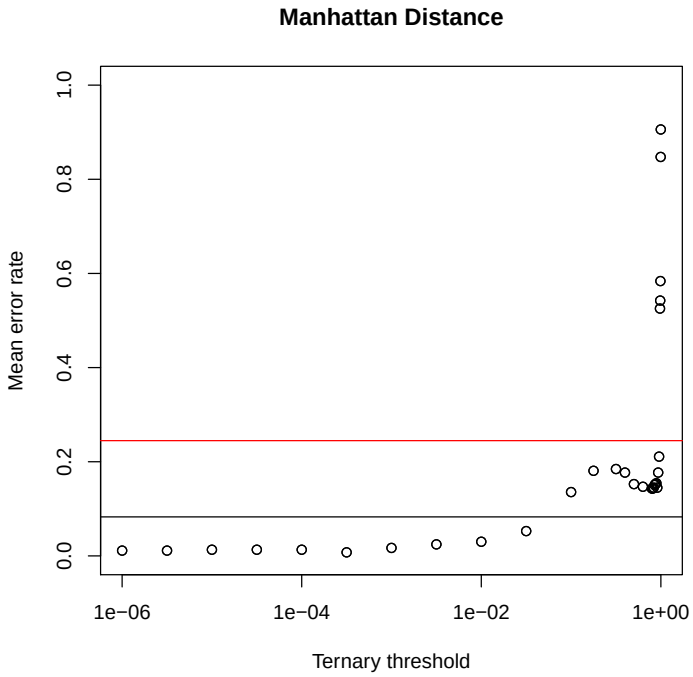

Figure 6: Plot of the error rate based on a leave-one-out cross-validation over the range of thresholds tested for the aggregated dataset based on the Manhattan distance. The black line indicates the ratio for the unthresholded POE matrix, and the red for the barcode. Error rates are 0.017, 0.083, 0.245, and 0.642.

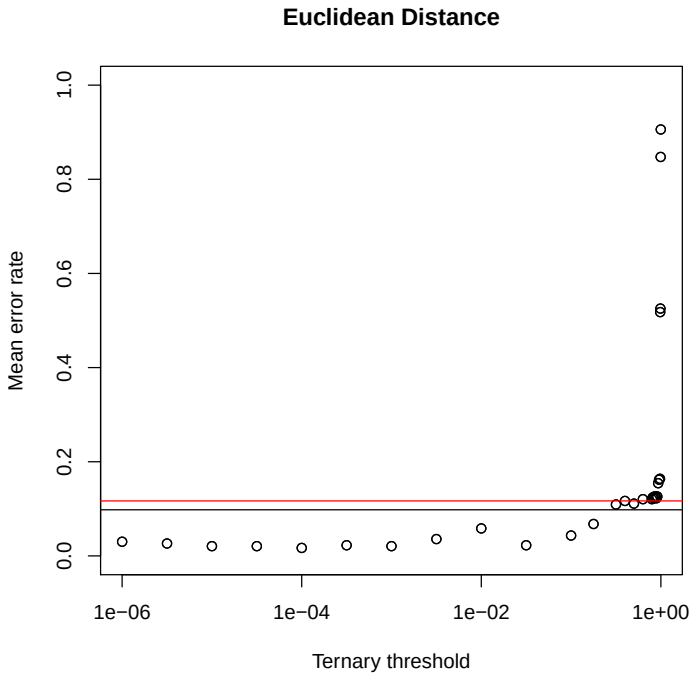

Figure 7: Plot of the error rate based on a leave-one-out cross-validation over the range of thresholds tested for the aggregated dataset based on the Euclidean distance. The black line indicates the ratio for the unthresholded POE matrix, and the red for the barcode. Error rates are 0.021, 0.098, 0.117, and 0.629.

```
> precisionRecall<-function(
+   testMatrix, recallMatrix, testClass, recallClass, method = "manhattan")
+   # Calculate precision and recall based on rank of distance pairs
+   {
+     # check values
+     if (!(all.equal(colnames(testMatrix), names(testClass)))
+         ) stop("Check test matrix and class should names match")
+     if (!(all.equal(colnames(recallMatrix), names(recallClass)))
+         ) stop("Recall matrix and class names should match")
+     # define an object to hold the ranked lists
+     ordered<-vector("list", ncol(testMatrix))
+     for (i in 1:ncol(testMatrix)){
+       # rank by distance or correlation
+       if (method == "manhattan"){
+       ordered[[i]] <- sort(colSums(abs(apply(recallMatrix,
+                           2,
+                           function(x){x - testMatrix[,i, drop = FALSE]}
+                           ))))
+         }
+       else if (method == "euclidean"){
+       ordered[[i]] <- sort(colSums(abs(apply(recallMatrix,
+                           2,
+                           function(x){(x - testMatrix[,i, drop = FALSE])^2}
+                           ))))
+         }
+       else if (method == "spearman"){
+       ordered[[i]] <- sort(cor(testMatrix[,i, drop = FALSE], recallMatrix,
+                                method = "spearman")[1,],
+                       decreasing = TRUE)
+         }
+
+       # set names as correct or incorrectly matched
+       names(ordered[[i]])<-(recallClass[
+                               names(ordered[[i]])
+                                       ] == testClass[i])
+       }
+   # compile list from all samples
+   if (method %in% c("manhattan", "euclidean")){
+     ordered.all<-sort(unlist(ordered))
+   }
+   else if (method == "spearman"){
+     ordered.all<-sort(unlist(ordered), decreasing = TRUE)
+   }
+   logical.ordered<-as.logical(names(ordered.all))
+   # define precision and recall
+   recall<-cumsum(logical.ordered)/sum(logical.ordered)
```

```
+   precision<-cumsum(logical.ordered)/(1:length(logical.ordered))
+   precision.recall<-data.frame(recall = recall, precision = precision)
+   average.precision<-mean(precision[logical.ordered])
+   attr(precision.recall, "Av.precision") <- average.precision
+   return(precision.recall)
+   }
```

Now we can use this to calculate the PR across the range of thresholds.

```
> precision.recall<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   human.threshold<-ternaryThreshold(human.matrix, threshold)
+   GPL1261.threshold<-ternaryThreshold(GPL1261.matrix, threshold)
+   precisionRecall(testMatrix = GPL1261.threshold,
+                   recallMatrix = human.threshold,
+                   testClass = mouse.class,
+                   recallClass = human.class
+                   )
+   })
```

```
> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^", n[i], sep = ""))
+   lines(precision.recall[[i]])
+   }
> par(op)
```

A precision-recall curve can also be constructed for the barcode matrix

```
> precision.recall.barcode <- precisionRecall(
+                   testMatrix = GPL1261.barcode.matrix,
+                   recallMatrix = human.barcode.matrix,
+                   testClass = mouse.class,
+                   recallClass = human.class
+                   )
```

```
> plot.new()
> axis(1, seq(0,1,0.2))
> axis(2, seq(0,1,0.2))
> title(main = "precision-recall for barcode and fingerprint", xlab = "recall", ylab = "
```

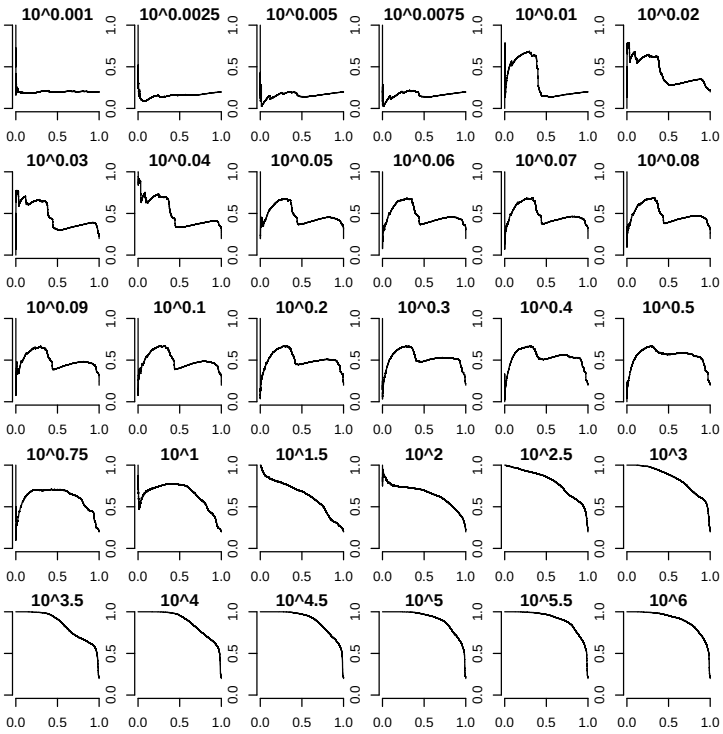

Figure 8: Precision-recall curves for matching mouse to human tissue data

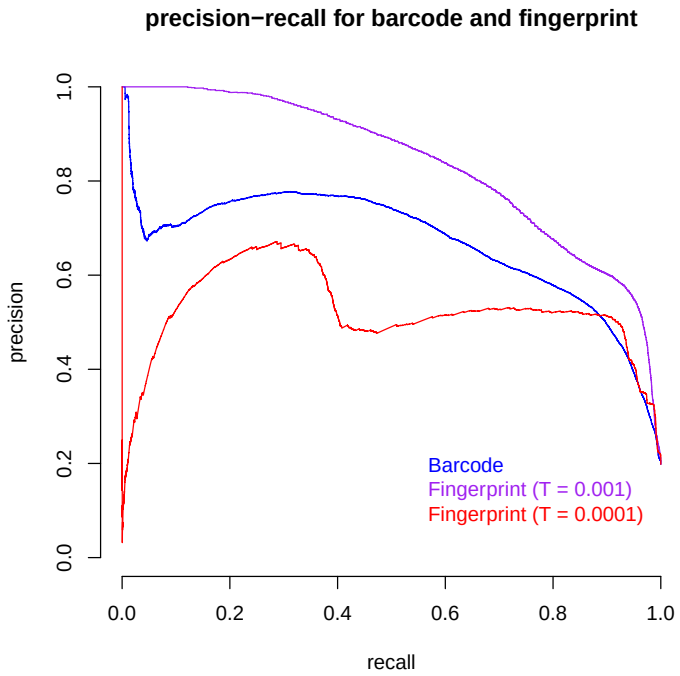

Figure 9: Precision-recall curves for the barcode and fingerprint

```
> lines(precision.recall.barcode, col = "blue")
> lines(precision.recall[[which(N == 3)]], col = "purple")
> lines(precision.recall[[16]], col = "red")
> legend(0.5,0.25, c("Barcode", "Fingerprint (T = 0.001)", "Fingerprint (T = 0.0001)"),
+         text.col = c("blue", "purple", "red"), bty= "n")
> par(op)
```

Ranking all possible distances between the samples may not be the best approach due to different inter vs intra tissue distances. We can plot a PCA of the data from `Figure 3` to provide a view on this.

```
> GPL570.GPL96.GPL1261.0.001<-cbind(
+                            ternaryThreshold(GPL570.matrix, 0.001),
+                            ternaryThreshold(GPL96.matrix, 0.001),
+                            ternaryThreshold(GPL1261.matrix, 0.001)
+                            )
> GPL570.GPL96.GPL1261.0.0001<-cbind(
+                            ternaryThreshold(GPL570.matrix, 0.0001),
+                            ternaryThreshold(GPL96.matrix, 0.0001),
```

```
+                           ternaryThreshold(GPL1261.matrix, 0.0001)
+                           )
> GPL570.GPL96.GPL1261.barcode<-barcode.matrix[,
+                              c(GPL570$DB_ID, GPL96$DB_ID, GPL1261$DB_ID)
+                              ]
> pc.0.001<-prcomp(t(GPL570.GPL96.GPL1261.0.001))
> pc.0.0001<-prcomp(t(GPL570.GPL96.GPL1261.0.0001))
> pc.barcode<-prcomp(t(GPL570.GPL96.GPL1261.barcode))


> pc.colors<-rainbow(6)[as.factor(c(GPL570$Tissue, GPL96$Tissue, GPL1261$Tissue))]
> pc.pch<-c(1,2,3)[as.factor(c(GPL570$Platform, GPL96$Platform, GPL1261$Platform))]
> par(mfcol = c(2,2), cex = 0.5)
> plot(pc.0.001$x, col = pc.colors, pch = pc.pch,
+       main = "Fingerprint T = 0.001")
> plot(pc.0.0001$x, col = pc.colors, pch = pc.pch,
+       main = "Fingerprint T = 0.0001")
> plot(pc.barcode$x, col = pc.colors, pch = pc.pch,
+       main = "Barcode")
> plot.new()
> legend("left", levels(as.factor(c(GPL570$Tissue, GPL96$Tissue, GPL1261$Tissue))),
+       text.col = rainbow(6)[1:6], cex = 2, bty = "n")
> legend("right",legend = levels(as.factor(c(GPL570$Platform, GPL96$Platform, GPL1261$Pl
+       pch = c(1,2,3), cex = 2, bty = "n")
```

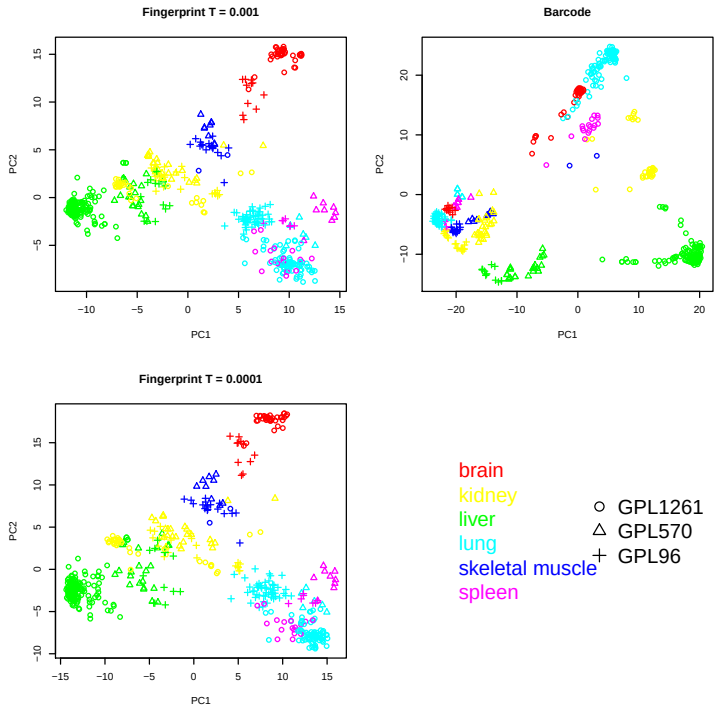

Figure 10: Plots of the two most significant principal components for tissue gene expression data processed using the barcode and fingerprint. The barcode provides marginally better clustering of samples within a platform while the fingerprint more effectively groups samples across platforms

1.4.2 Approach 2

An alternative approach is to construct PR curves for each tissue individually. This can then be combined (using a weighted mean if necessary) to yield an overall PR curve.

```
> PR.tissues<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   human.threshold<-ternaryThreshold(human.matrix, threshold)
+   GPL1261.threshold<-ternaryThreshold(GPL1261.matrix, threshold)
+   precision.recall.tissue<-vector("list", length(levels(as.factor(GPL1261$Tissue))))
+   names(precision.recall.tissue)<-levels(as.factor(GPL1261$Tissue))
+   for (i in 1:length(levels(as.factor(GPL1261$Tissue)))){
+     tissue<-levels(as.factor(GPL1261$Tissue))[i]
+     precision.recall.tissue[[i]] <- precisionRecall(
+       testMatrix = GPL1261.threshold[,GPL1261$Tissue == tissue],
+       recallMatrix = human.threshold,
+       testClass = mouse.class[GPL1261$Tissue == tissue],
+       recallClass = human.class
+       )
+     }
+   precision.recall.tissue
+   })
```

We can attempt to combine the tissues by taking an average of the PR curves. As each tissue has a different number of samples we need to interpollate the precision at a defined set of recall values. This will be considered as a next step.
Another way to view the problem is to construct a PR curve for each sample individually and then combine these curves. Need to interpollate so that we can aggregate points with the same recall value.

```
> PR.samples<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   human.threshold<-ternaryThreshold(human.matrix, threshold)
+   GPL1261.threshold<-ternaryThreshold(GPL1261.matrix, threshold)
+   precision.recall.sample<-matrix(nrow = 100, ncol = ncol(GPL1261.threshold))
+   colnames(precision.recall.sample)<-colnames(GPL1261.threshold)
+   for (i in 1:ncol(GPL1261.threshold)){
+      PR <- precisionRecall(
+       testMatrix = GPL1261.threshold[,i, drop = FALSE],
+       recallMatrix = human.threshold,
+       testClass = mouse.class[i],
+       recallClass = human.class
+       )
```

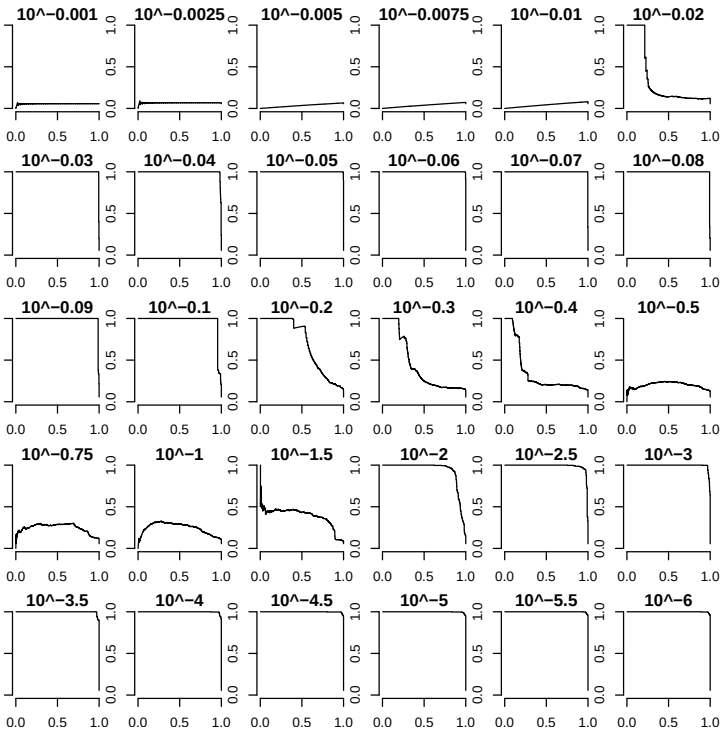

Figure 11: Precision-recall curves for matching mouse to human *brain* data

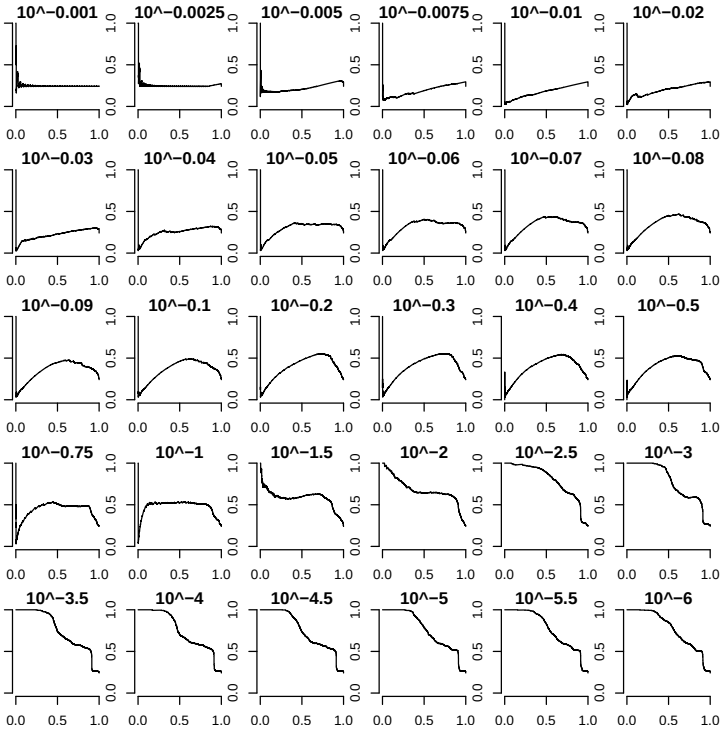

Figure 12: Precision-recall curves for matching mouse to human *kidney* data

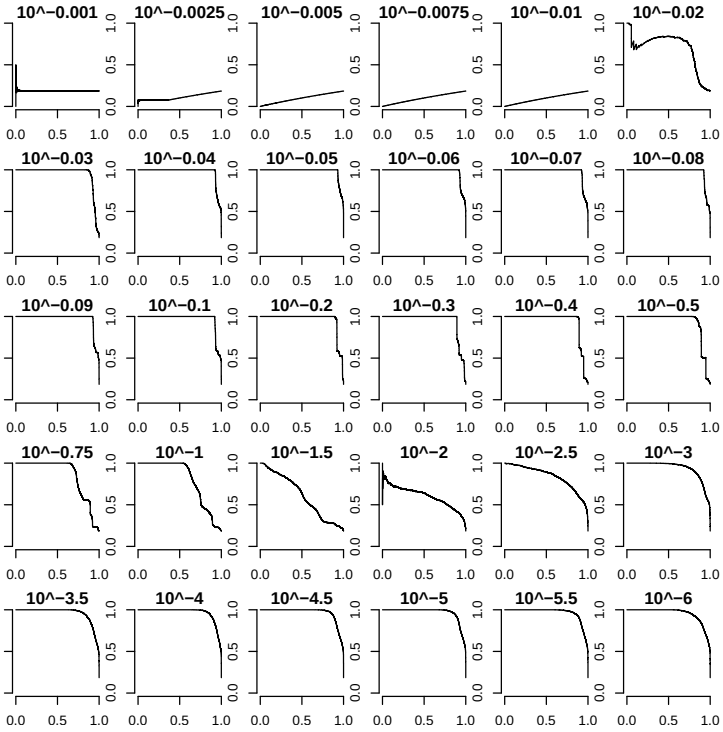

Figure 13: Precision-recall curves for matching mouse to human *liver* data

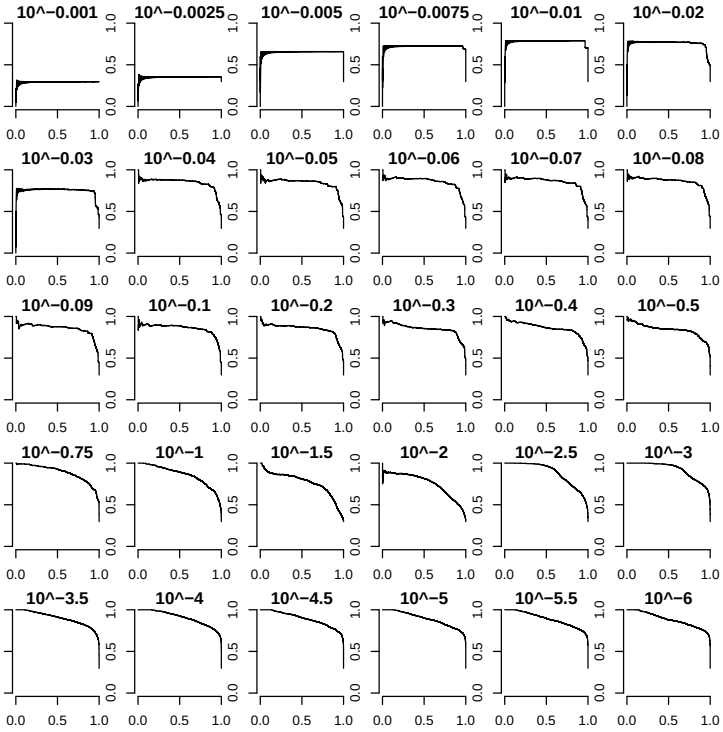

Figure 14: Precision-recall curves for matching mouse to human *lung* data

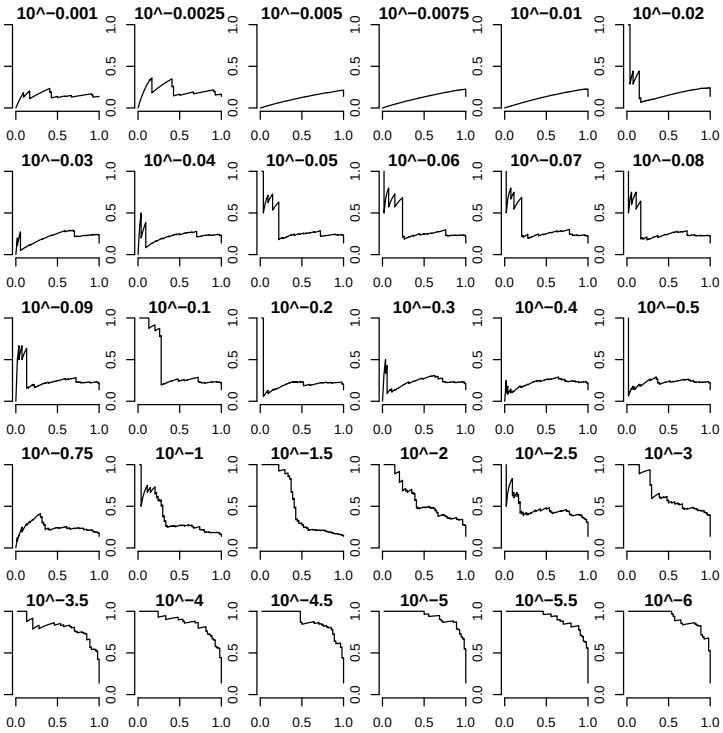

Figure 15: Precision-recall curves for matching mouse to human *skeletalmuscle* data

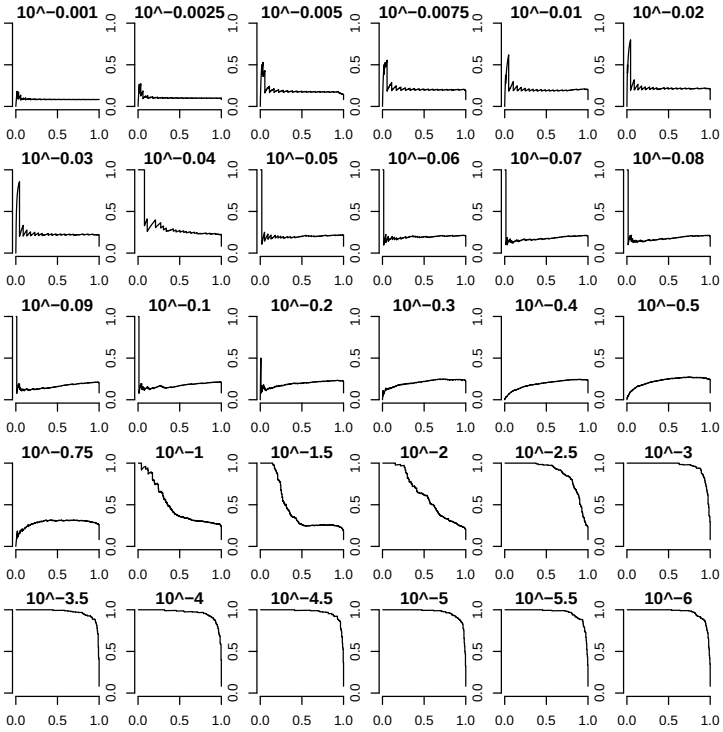

Figure 16: Precision-recall curves for matching mouse to human *spleen* data

```
+     precision.recall.sample[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+     }
+   rowMeans(precision.recall.sample)
+   })
```

Repeating this analysis for the barcode allows a direct comparison between the methods

```
> PR.samples.barcode<-matrix(nrow = 100, ncol = ncol(GPL1261.barcode.matrix))
> colnames(PR.samples.barcode)<-colnames(GPL1261.barcode.matrix)
> for (i in 1:ncol(GPL1261.barcode.matrix)){
+  PR <- precisionRecall(
+   testMatrix = GPL1261.barcode.matrix[,i, drop = FALSE],
+   recallMatrix = human.barcode.matrix,
+   testClass = mouse.class[i],
+   recallClass = human.class
+   )
+   PR.samples.barcode[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+     }
> PR.samples.barcode<-rowMeans(PR.samples.barcode)
```

1.4.3 Comparison to randomly constructed genesets

The fingerprint was establish based on the hypothesis that aggregating genes based on pathways provides a means to integrate 'expert' prior biological knowledge with gene expression data. To test that this knownledge contributes to the success of the fingerprint we have constructed a 'random' fingerprint based on gene sets produced by randomly sampling the pathway fingeprint gene sets. The size distribution has been retained, as has the overall list and frequency of gene IDs.

```
> PR.samples.random<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   human.threshold<-ternaryThreshold(human.random.matrix, threshold)
+   GPL1261.threshold<-ternaryThreshold(GPL1261.random.matrix, threshold)
+   precision.recall.sample<-matrix(nrow = 100, ncol = ncol(GPL1261.threshold))
```

```
> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^-", n[i], sep = ""))
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples[[i]])
+   }
>   par(op)
```

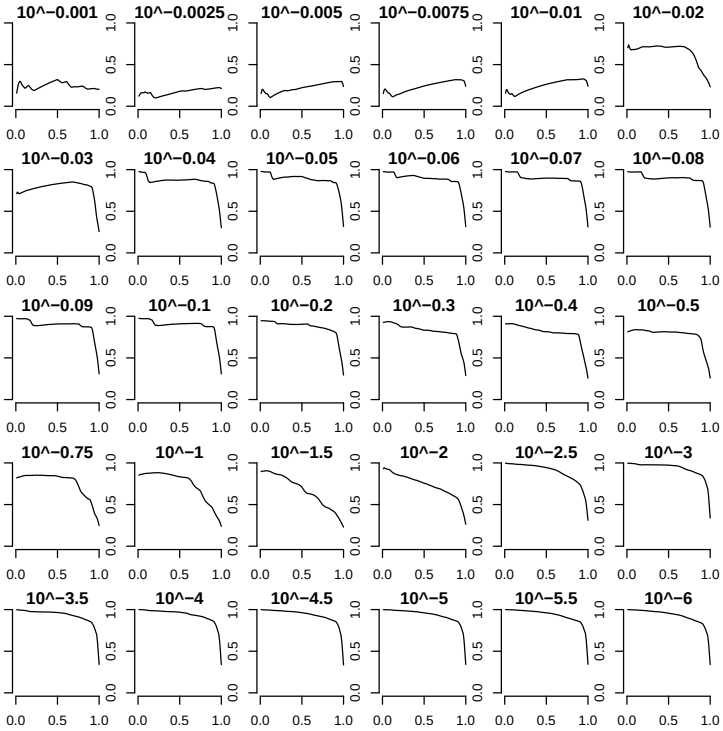

Figure 17: Precision-recall curves for aggregated mouse to human tissues data

```
> plot.new()
> axis(1, seq(0,1,0.2))
> axis(2, seq(0,1,0.2))
> title(xlab = "recall", ylab = "precision")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.barcode, col = "blue")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples[[which(N == 3)]], col = "purple")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples[[16]], col = "red")
> legend(0.5,0.25, c("Barcode", "Fingerprint (T = 0.001)", "Fingerprint (T = 0.0001)"),
+         text.col = c("blue", "purple", "red"), bty= "n")
```

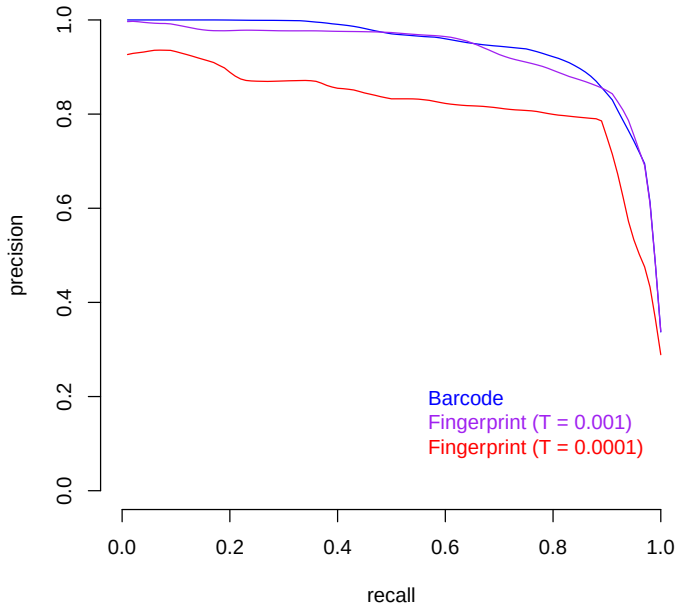

Figure 18: Aggregated sample-based precision-recall curves for mouse to human tissue data

```
+   colnames(precision.recall.sample)<-colnames(GPL1261.threshold)
+   for (i in 1:ncol(GPL1261.threshold)){
+      PR <- precisionRecall(
+       testMatrix = GPL1261.threshold[,i, drop = FALSE],
+       recallMatrix = human.threshold,
+       testClass = mouse.class[i],
+       recallClass = human.class
+       )
+     precision.recall.sample[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+     }
+   rowMeans(precision.recall.sample)
+   })
```

```
> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^-", n[i], sep = ""))
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples.random[[i]], col = "blue")
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples[[i]], col = "red")
+   }
>   par(op)
```

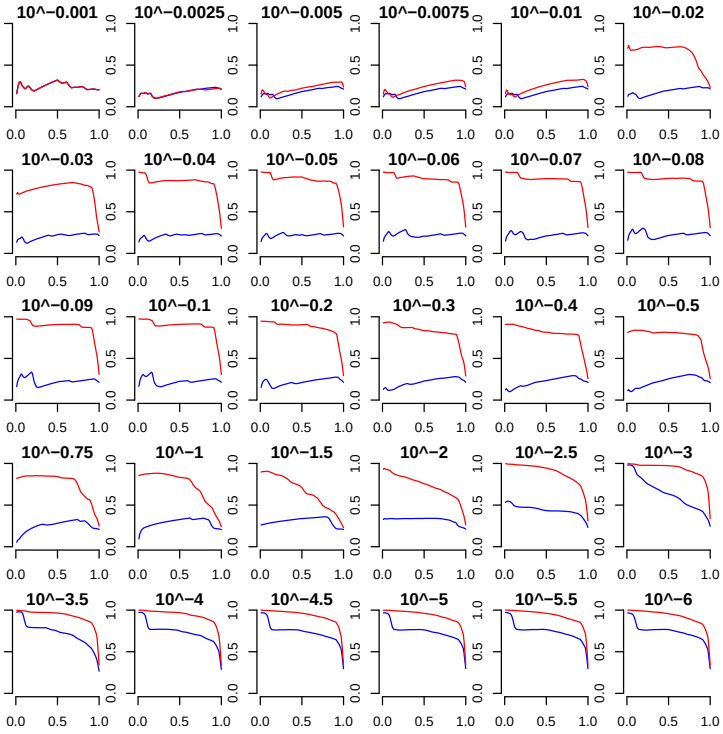

Figure 19: Precision-recall curves for aggregated mouse to human tissues data based on a fingerprint build using pathways and functional interaction modules (red) or randomly constructed gene sets (blue).

1.4.4 Approach 3

Here we will group all samples and construct PR curves for each one, irrespective of it's source. This will test the ability of the fingerprint to classify samples both within and between platforms

```
> PR.samples.all<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.matrix, threshold)
+   pr.sample.all<-matrix(nrow = 100, ncol = ncol(full.threshold))
+   av.precision<-vector("numeric", ncol(full.threshold))
+   colnames(pr.sample.all)<-colnames(full.threshold)
+   for (i in 1:ncol(full.threshold)){
+      PR <- precisionRecall(
+       testMatrix = full.threshold[,i, drop = FALSE],
+       recallMatrix = full.threshold[,-i],
+       testClass = full.class[i],
+       recallClass = full.class[-i]
+       )
+     pr.sample.all[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+     av.precision[i]<-attr(PR, "Av.precision")
+     }
+   interpol <- list(
+     PR = rowMeans(pr.sample.all),
+     MAP = mean(av.precision)
+     )
+   })


> full.random.matrix<-cbind(human.random.matrix, GPL1261.random.matrix)
> PR.samples.random.all<-invisible(foreach (j = 1:30) %dopar% {
+   n <- N[j]
+   threshold<-10^(-n)
+   full.threshold<-ternaryThreshold(full.random.matrix, threshold)
+   pr.sample.all<-matrix(nrow = 100, ncol = ncol(full.threshold))
+   av.precision<-vector("numeric", ncol(full.threshold))
+   colnames(pr.sample.all)<-colnames(full.threshold)
+   for (i in 1:ncol(full.threshold)){
+      PR <- precisionRecall(
+       testMatrix = full.threshold[,i, drop = FALSE],
+       recallMatrix = full.threshold[,-i],
```

```
+       testClass = full.class[i],
+       recallClass = full.class[-i]
+       )
+     pr.sample.all[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+     av.precision[i]<-attr(PR, "Av.precision")
+     }
+   interpol <- list(
+     PR = rowMeans(pr.sample.all),
+     MAP = mean(av.precision)
+     )
+   })

> PR.samples.barcode.all<-matrix(nrow = 100, ncol = ncol(full.barcode.matrix))
> colnames(PR.samples.barcode.all)<-colnames(full.barcode.matrix)
> barcode.av.precision<-vector("numeric", ncol(full.barcode.matrix))
> for (i in 1:ncol(full.barcode.matrix)){
+  PR <- precisionRecall(
+   testMatrix = full.barcode.matrix[,i, drop = FALSE],
+   recallMatrix = full.barcode.matrix[,-i],
+   testClass = full.class[i],
+   recallClass = full.class[-i]
+   )
+   PR.samples.barcode.all[,i]<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+   barcode.av.precision[i]<-attr(PR, "Av.precision")
+   }
> PR.samples.barcode.all<-rowMeans(PR.samples.barcode.all)
> attr(PR.samples.barcode.all, "mean.av.precision") <- mean(barcode.av.precision)

> PR.samples.spearman.all<-invisible(foreach (i = 1:ncol(barcode.ranks)) %dopar% {
+  PR <- precisionRecall(
+   testMatrix = barcode.ranks[,i, drop = FALSE],
+   recallMatrix = barcode.ranks[,-i],
+   testClass = full.class[i],
+   recallClass = full.class[-i],
+   method = "spearman"
```

```
+   )
+   PR.samples.spearman<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+   interpol<-list(
+     PR = PR.samples.spearman,
+     AP = attr(PR, "Av.precision")
+     )
+   })
> spearman.MAP<-mean(unlist(
+   lapply(PR.samples.spearman.all, function(x){x[["AP"]]}))
+                     )
> PR.samples.spearman.all<-rowMeans(as.data.frame(
+   lapply(PR.samples.spearman.all, function(x){x[["PR"]]})
+   ))

> PR.samples.unthresholded.all<-invisible(foreach (i = 1:ncol(full.matrix)) %dopar% {
+  PR <- precisionRecall(
+   testMatrix = full.matrix[,i, drop = FALSE],
+   recallMatrix = full.matrix[,-i],
+   testClass = full.class[i],
+   recallClass = full.class[-i],
+   method = "euclidean"
+   )
+   PR.samples.unthresholded<-approx(
+                 x = PR$recall,
+                 y = PR$precision,
+                 xout = seq(0.01, 1, 0.01),
+                 rule = 2
+                 ,ties = "mean")$y
+
+     interpol<-list(
+       PR = PR.samples.unthresholded,
+       AP = attr(PR, "Av.precision")
+       )
+   })
> unthresholded.MAP<-mean(unlist(
+   lapply(PR.samples.unthresholded.all, function(x){x[["AP"]]}))
+                     )
> PR.samples.unthresholded.all<-rowMeans(as.data.frame(
+   lapply(PR.samples.unthresholded.all, function(x){x[["PR"]]})
+   ))
```

```
> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^-", n[i], sep = ""))
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[i]]$PR)
+   }
>   par(op)
```

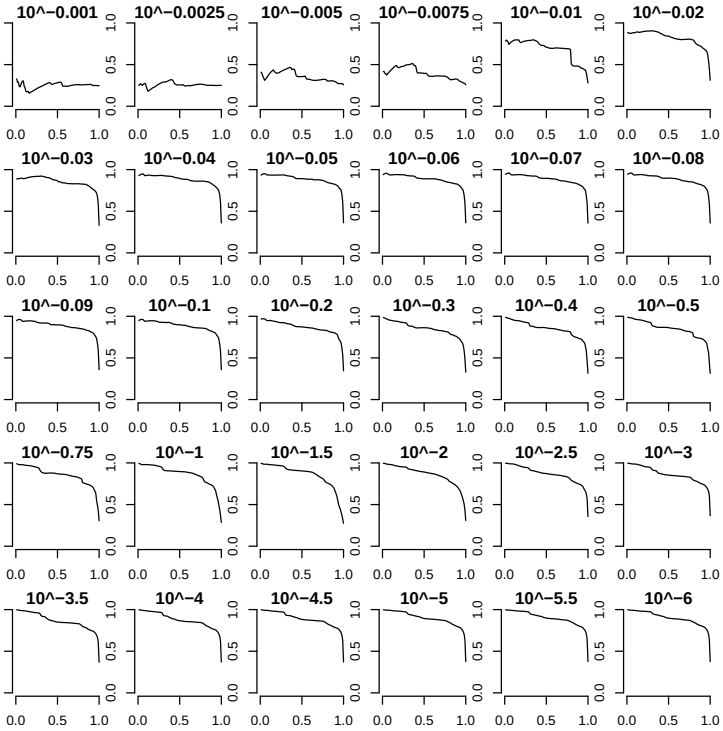

Figure 20: Precision-recall curves for aggregated data - approach 3

```
> plot.new()
> axis(1, seq(0,1,0.2))
> axis(2, seq(0,1,0.2))
> title(xlab = "recall", ylab = "precision")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.barcode.all, col = "blue")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[which(N == 3)]]$PR, col = "purple")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[which(N == 4)]]$PR, col = "pink")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[which(N == 5)]]$PR, col = "red")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.spearman.all, col = "green")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.unthresholded.all, col = "black")
> legend(0.5,0.35, c("Barcode", "Fingerprint (T = 0.001)",
+                    "Fingerprint (T = 0.0001)", "Fingerprint (T = 0.00001)",
+                    "Spearman", "Unthresholded"),
+          text.col = c("blue", "purple", "pink", "red", "green", "black"), bty= "n")
```

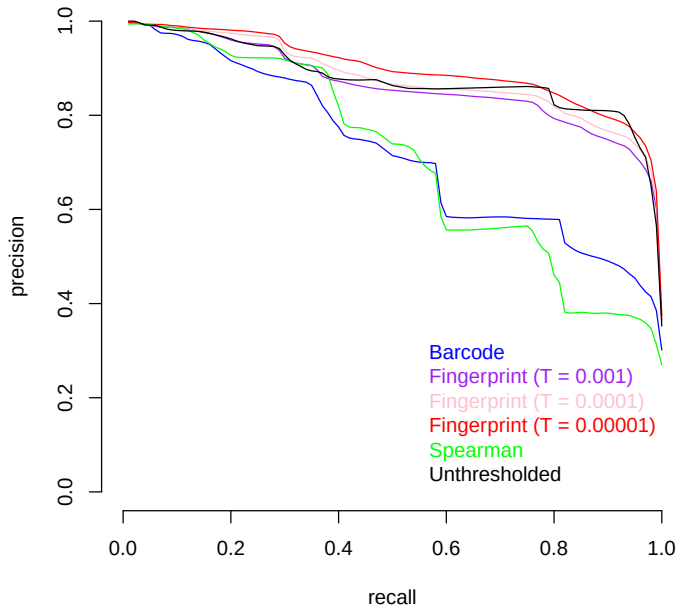

Figure 21: Precision-recall curves for aggregated data - approach 3

```
> plot(x = 10^(-N),
+      y = unlist(lapply(
+        PR.samples.all, function(x){x[["MAP"]]}
+        )),
+      log = "x",
+      xlab = "Ternary threshold",
+      ylab = "Mean average precision",
+      ylim = c(0, 1)
+      )
> points(x = 10^(-N),
+      y = unlist(lapply(
+        PR.samples.random.all, function(x){x[["MAP"]]}
+        )), col = "blue")
> abline(h = attr(PR.samples.barcode.all, "mean.av.precision"), col = "blue")
> abline(h = spearman.MAP, col = "green")
> abline(h = unthresholded.MAP, col = "red")
```

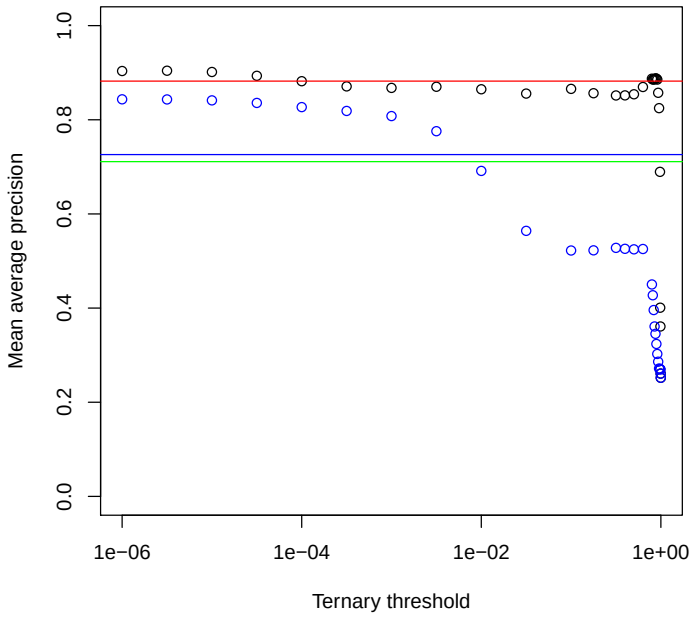

Figure 22: Plot of the mean average precision over the range of thresholds tested for the aggregated dataset (approach 3) for the pathway fingerprint (black circles) and a fingerprint build on random gene sets of equivalent size distribution (blue circles). Solid lines indicate the mean average precision for Barcode (blue), Spearman correlation (green) and the unthresholded fingerprint (red)

```
> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^-", n[i], sep = ""))
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples.random.all[[i]]$PR, col = "blue")
+   lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[i]]$PR, col = "red")
+   }
>   par(op)
```

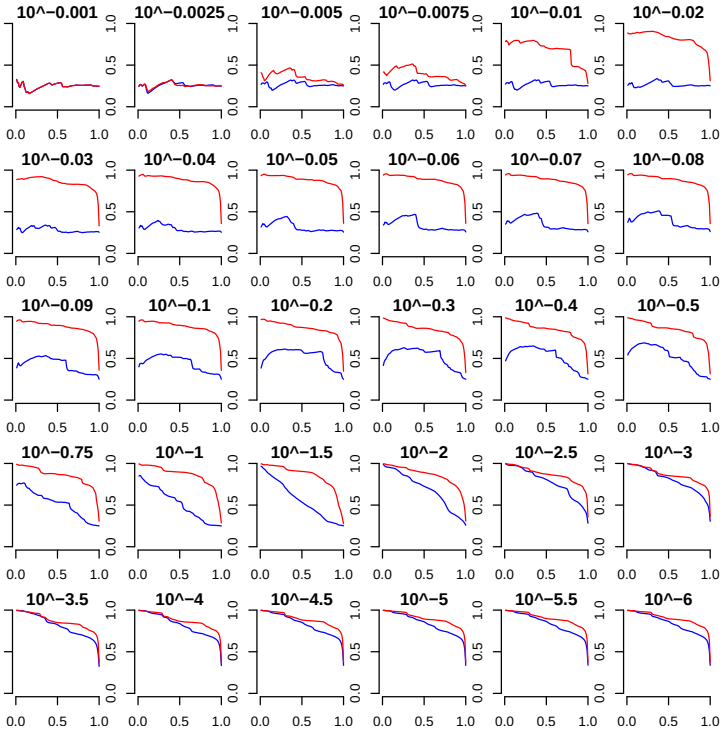

Figure 23: Precision-recall curves for aggregated mouse to human tissues data based on a fingerprint build using pathways and functional interaction modules (red) or randomly constructed gene sets (blue) by approach 3.

2 Figure for manuscript

For the pathway fingerprinting mauscript we require a figure comparing the precision-recall for the fingerprint, barcode, spearman and random genesets.

3 Cross-species hematopoesis profiling

We are also interested to examine how well the fingerprint can group hematopeoitic samples, cross-species. Matched cell-type data has already been collated from *Novershtern et al. Densely Interconnected Transcriptional Circuits Control Cell States in Human Hematopoiesis. Cell (2011)* and *Chambers et al. Hematopoietic fingerprints: an expression database of stem cells and their progeny. Cell Stem Cell (2007) vol. 1 (5) pp. 578-91.*

```
> load(
+   "/home/galtschu2/Documents/Projects/Fingerprinting/data/sq_.POE.matrix.2011-07-27.RD
+   )
> bloodSet<-read.delim("/data/shared/Fingerprint/curatedCellTypes/bloodTypesHumanMouse.t
> bloodSet.POE<-POE.matrix[,bloodSet$gsm]
> rm(POE.matrix)
> head(bloodSet)
```

```
       gsm                                                title       type
1 GSM149579 Long-term hematopoietic stem cells LT-HSC_SPKSL_1     LT_HSC
2 GSM149580 Long-term hematopoietic stem cells LT-HSC_SPKSL_2     LT_HSC
3 GSM149581                                     NK cells NK_1         NK
4 GSM149582                                     NK cells NK_2         NK
5 GSM149583                        CD4+ naive T-cells t4n_1 CD4+_naive
6 GSM149584                        CD4+ naive T-cells t4n_2 CD4+_naive
  species platform
1   mouse  GPL1261
2   mouse  GPL1261
3   mouse  GPL1261
4   mouse  GPL1261
5   mouse  GPL1261
6   mouse  GPL1261
```

This data contains human and mouse data

```
> blood.human<-bloodSet[bloodSet$species == "human",]
> blood.mouse<-bloodSet[bloodSet$species == "mouse",]
```

```
> err.blood <- matrix(nrow = 30, ncol=2)
> for (j in 1:30){
+   blood.mouse$predictedCellType<-NA
+   n <- N[j]
```

```
> plot.new()
> axis(1, seq(0,1,0.2))
> axis(2, seq(0,1,0.2))
> title(xlab = "recall", ylab = "precision")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.barcode.all, col = "blue")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.all[[which(N == 3)]]$PR, col = "red")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.spearman.all, col = "green")
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.random.all[[which(N == 3)]]$PR, col = "bl
> lines(x = seq(0.01, 1, 0.01), y = PR.samples.unthresholded.all, col = "red", lty = 2)
```

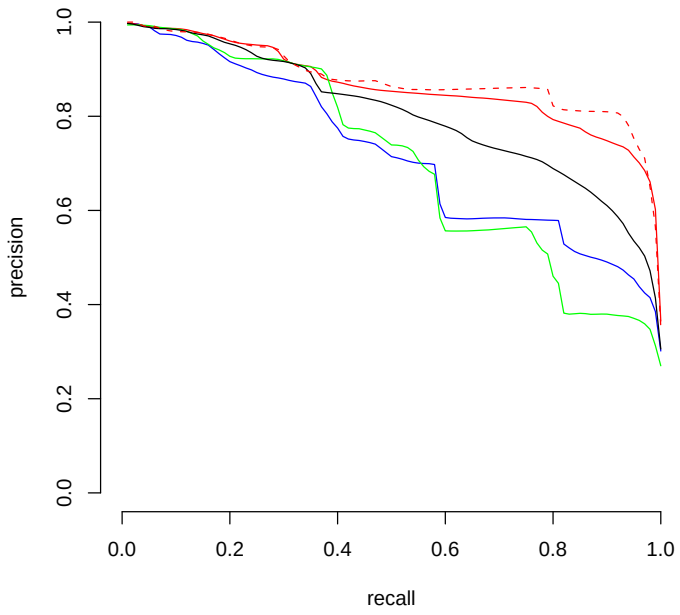

Figure 24: Precision-recall curves for cross-species tissue data for the Pathway Fingerprint (red, MAP = 0.868), unthresholded POE scores (dashed red, MAP = 0.882), a fingerprint based on random genesets (black, MAP = 0.808), the gene expression barcode (blue, MAP = 0.726), and spearman correlation (green, , MAP = 0.711).

```
+   threshold<-10^(-n)
+   blood.mouse.threshold<-ternaryThreshold(bloodSet.POE[,blood.mouse$gsm], threshold)
+   blood.human.threshold<-ternaryThreshold(bloodSet.POE[,blood.human$gsm], threshold)
+   for (i in 1:nrow(blood.mouse)){
+     mouse.fingerprint<-blood.mouse.threshold[,blood.mouse$gsm[i], drop = FALSE]
+     ordered <- sort(colSums(abs(apply(blood.human.threshold, 2, function(x){x - mouse.
+     blood.mouse$predictedCellType[i]<-blood.human$type[blood.human$gsm == names(ordere
+     }
+   Actual <- as.factor(blood.mouse$type)
+   Predicted<-factor(blood.mouse$predictedCellType, levels = levels(Actual))
+   m<-table(Actual = Actual, Predicted = Predicted)
+   err.blood[j,] <- c(threshold, 1 - sum(diag(m)) / sum(m))
+ }
```

Again, the confusion matrix is used to calculate the error rate, for example

```
> m
                      Predicted
Actual                 B-Cell CD4+_activated CD4+_naive CD8+_activated
  B-Cell                    2              0          0              0
  CD4+_activated            0              0          2              0
  CD4+_naive                2              0          0              0
  CD8+_activated            0              0          2              0
  CD8+_naive                2              0          0              0
  Granulocyte               0              0          0              0
  LT_HSC                    1              0          1              0
  Monocyte                  0              0          0              0
  NK                        2              0          0              0
  Nucleated Erythrocytes    0              0          0              0
                      Predicted
Actual                 CD8+_naive Granulocyte LT_HSC Monocyte NK
  B-Cell                        0           0      0        0  0
  CD4+_activated                0           0      0        0  0
  CD4+_naive                    0           0      0        0  0
  CD8+_activated                0           0      0        0  0
  CD8+_naive                    0           0      0        0  0
  Granulocyte                   0           2      0        0  0
  LT_HSC                        0           0      0        0  0
  Monocyte                      0           2      0        0  0
  NK                            0           0      0        0  0
  Nucleated Erythrocytes        0           0      0        0  0
                      Predicted
Actual                 Nucleated Erythrocytes
  B-Cell                                    0
  CD4+_activated                            0
  CD4+_naive                                0
```

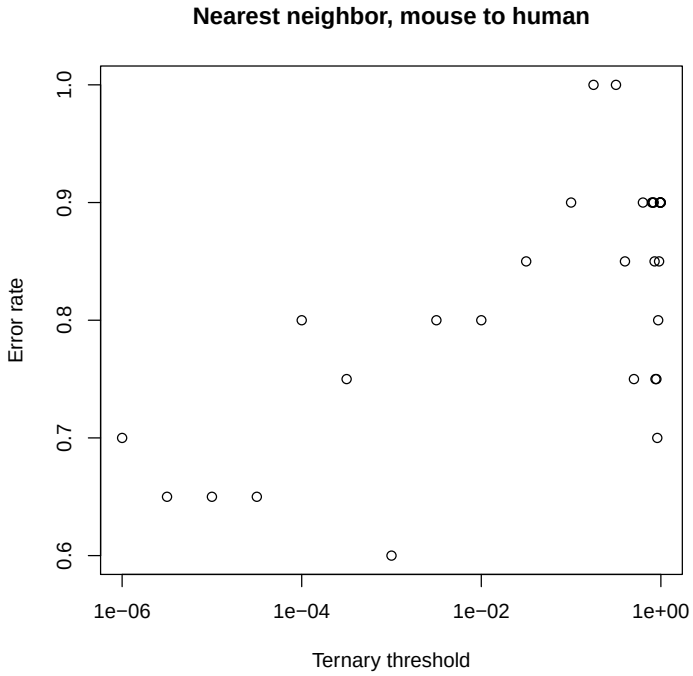

Figure 25: Error rate vs fingerprint threshold

```
CD8+_activated                  0
CD8+_naive                      0
Granulocyte                     0
LT_HSC                          0
Monocyte                        0
NK                              0
Nucleated Erythrocytes          2
```

gives an error rate of

```
> 1 - sum(diag(m)) / sum(m)
[1] 0.7
```

```
> plot(err.blood, log = "x", xlab = "Ternary threshold", ylab = "Error rate",
+     main = "Nearest neighbor, mouse to human")
```

An alternative approach is to try to define precision-recall curves. For this we will calculate the distance from each mouse array to all of the human arrays (or

vice-versa). This can be used to construct a list of all the pairs, ranked by their distance.

```
> precision.recall<-vector("list", 30)
> for (j in 1:30){
+   blood.mouse$predictedCellType<-NA
+   n <- N[j]
+   threshold<-10^(-n)
+   blood.mouse.threshold<-ternaryThreshold(bloodSet.POE[,blood.mouse$gsm], threshold)
+   blood.human.threshold<-ternaryThreshold(bloodSet.POE[,blood.human$gsm], threshold)
+   ordered<-vector("list", nrow(blood.mouse))
+   for (i in 1:nrow(blood.mouse)){
+     # create mouse fingerprint
+     mouse.fingerprint<-blood.mouse.threshold[,blood.mouse$gsm[i], drop = FALSE]
+     # Order matched arrays
+     ordered[[i]] <- sort(colSums(abs(apply(blood.human.threshold, 2, function(x){x - m
+     # Define matched arrays as correct or incorrect
+     names(ordered[[i]])<-(blood.human$type[match(names(ordered[[i]]), blood.human$gsm)
+     }
+   ordered.all<-sort(unlist(ordered))
+   x<-1:length(ordered.all)
+   recall<-sapply(x, function(x){
+                 sum(as.logical(names(ordered.all))[1:x])/sum(as.logical(names(ordered.
+                 })
+   precision<-sapply(x, function(x){
+                   sum(as.logical(names(ordered.all))[1:x])/x
+                   })
+   precision.recall[[j]]<-data.frame(recall = recall, precision = precision)
+   }


> op<-par(mfrow = c(1,1), pty = "s")
> par(mfrow = c(5,6), mar = c(1,1,1,1))
> n <- N
> for (i in 1:30){
+   plot.new()
+   axis(1, seq(0,1,0.5))
+   axis(2, seq(0,1,0.5))
+   title(paste("10^-", n[i], sep = ""))
+   lines(precision.recall[[i]])
+   }
> par(op)
```

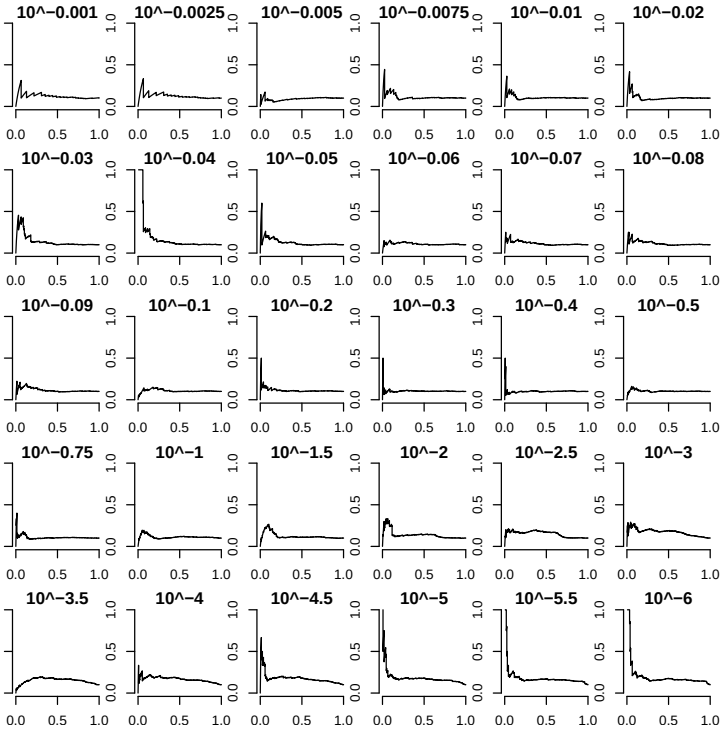

Figure 26: Precision-recall curves for matching mouse to human blood data